Reluctance to Trust

Sean Barnum, Cigital, Inc. [vita[3]]
Michael Gegick, Cigital, Inc. [vita[4]]

Copyright © 2005 Cigital, Inc.

2005-09-15 L4 / D/P, L[5]

Developers should assume that the environment in which their system resides is insecure. Trust, whether it is in external systems, code, people, etc., should always be closely held and never loosely given. When building an application, software engineers should anticipate malformed input from unknown users. Even if users are known, they are susceptible to social engineering attacks, making them potential threats to a system. Also, no system is one hundred percent secure, so the interface between two systems should be secured. Minimizing the trust in other systems can increase the security of your application.

Detailed Description Excerpts

According to Viega and McGraw [Viega 02] in Chapter 5, "Guiding Principles for Software Security," in "Principle 9: Be Reluctant to Trust" from pages 111-112:[9]

> People commonly hide secrets in client code, assuming those secrets will be safe. The problem with putting secrets in client code is that talented end users will be able to abuse the client and steal all its secrets. Instead of making assumptions that need to hold true, you should be reluctant to extend trust. Servers should be designed not to trust clients, and vice versa, since both clients and servers get hacked. A reluctance to trust can help with compartmentalization.
>
> For example, while shrink-wrapped software can certainly help keep designs and implementations simple, how can any off-the-shelf component be trusted to be secure? Were the developers security experts? Even if they were well versed in software security, are they also infallible? There are hundreds of products from security vendors with gaping security holes. Ironically, many developers, architects and managers in the security tool business don't actually know very much about writing secure code themselves. Many security products introduce more risk than they address.
>
> Trust is often extended far too easily in the area of customer support. Social engineering attacks are thus easy to launch against unsuspecting customer support agents, who have a proclivity to trust since it makes their jobs easier.
>
> "Following the herd" has similar problems. Just because a particular security feature is an emerging standard doesn't mean it actually makes any sense. And even if competitors are not following good security practices, you should still consider good security practices yourself. For example, we often hear people deny a need to encrypt sensitive data because their competitors aren't encrypting their data. This argument will hold up only as long as customers are not hacked. Once they are, they will look to blame someone for not being duly diligent about security.
>
> Skepticism is always good, especially when it comes to security vendors. Security vendors all too often spread suspect or downright false data to sell their products. Most snake oil peddlers work by spreading FUD—Fear, Uncertainty and Doubt. Many common warning signs can help identify security quacks. One of our favorites is the advertising of "million bit keys" for a secret-key encryption algorithm. Mathematics tells us that 256 bits will likely be a big enough symmetric key to protect messages through the lifetime of the universe, assuming the algorithm using the key is of high quality. People advertising more know too little about the theory of cryptography to sell worthwhile security products. Before making a security buy decision, make sure to do lots of research. One good

3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/35-BSI.html (Barnum, Sean)
4. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/345-BSI.html (Gegick, Michael)
9. All rights reserved. It is reprinted with permission from Addison-Wesley Professional.

place to start is the "Snake Oil" FAQ, available at http://www.interhack.net/people/cmcurtin/snake-oil-faq.html.

Sometimes it is prudent not to trust even yourself. It is all too easy to be short sighted when it comes to your own ideas and your own code. While everyone wants to be perfect, it is often wise to admit that nobody is and periodically get some objective, high-quality outside eyes to review what you're doing.

One final point to remember is that trust is transitive. Once you dole out some trust, you often implicitly extend it to anyone the trusted entity may trust. For this reason, trusted programs should not invoke untrusted programs, ever. It is also good to be careful when determining whether a program should be trusted or not; see Chapter 12 [in *Building Secure Software*] for a complete discussion.

When you spread around trust, be careful.

According to Howard and LeBlanc [Howard 02] in Chapter 3, "Security Principles to Live By," in "Assume External Systems Are Insecure" from pages 63-64:

Assuming external systems are insecure is related to defense in depth—the assumption is actually one of your defenses. Consider any data you receive from a system you do not have complete control over to be insecure and a source of attack. This is especially important when accepting input from users. Until you can prove otherwise, all external stimuli have the potential to be an attack.

External servers can also be a potential point of attack. Clients can be redirected in a number of ways to the wrong server. As is covered in more depth in Chapter 15 [of *Writing Secure Code*], "Socket security," the DNS infrastructure we rely on to find the correct server is not very robust. When writing client-side code, do not make the assumption that you're only dealing with a well-behaved server.

Don't assume that your application will always communicate with an application that limits the commands a user can execute from the user interface or Web-based client portion of your application. Many server attacks take advantage of the ease of sending malicious data to the server by circumventing the client altogether. The same issue exists in the opposite direction, clients compromised by rogue servers.

We include two relevant principles from NIST [NIST 01] in Section 3.3, "IT Security Principles," from page 11:

Minimize the system elements to be trusted.

Security measures include people, operations, and technology. Where technology is used, hardware, firmware, and software should be designed and implemented so that a minimum number of system elements need to be trusted in order to maintain protection. Further, to ensure cost-effective and timely certification of system security features, it is important to minimize the amount of software and hardware expected to provide the most secure functions for the system.

From page 8:

Assume that external systems are insecure.

The term information domain arises from the practice of partitioning information resources according to access control, need, and levels of protection required. Organizations implement specific measures to enforce this partitioning and to provide for the deliberate flow of authorized information between information domains. An external domain is one that is not under your control.

In general, external systems should be considered insecure. Until an external domain has been deemed "trusted," system engineers, architects, and IT specialists should presume the security measures of an external system are different than those of a trusted internal system and design the system security features accordingly.

According to Bishop [Bishop 03] in Chapter 18, "Introduction to Assurance," in "Assurance and Trust" from pages 477-478:

In previous chapters we have used the terms *trusted system* and *secure system* without defining them precisely. When looked on as an absolute, creating a secure system is an ultimate, albeit unachievable, goal. As soon as we have figured out how to address one type of attack on a system, other types of attacks occur. In reality, we cannot yet build systems that are guaranteed to be secure or to remain secure over time. However, vendors frequently use the term "secure" in product names and product literature to refer to products and systems that have "some" security included in their design and implementation. The amount of security provided can vary from a few mechanisms to specific well-defined security requirements and well-implemented security mechanisms to those requirements. However, providing security requirements and functionality may not be sufficient to engender trust in the system.

Intuitively, trust is a belief or desire that a computer entity will do what it should to protect resources and be safe from attack. However, in the realm of computer security, trust has a very specific meaning. We will define trust in terms of a related concept.

Definition 18-1. An entity is trustworthy if there is sufficient credible evidence leading one to believe that the system will meet a set of given requirements. Trust is a measure of trustworthiness, relying on the evidence provided.

These definitions emphasize that calling something "trusted" or "trustworthy" does not make it so. Trust and trustworthiness in computer systems must be backed by concrete evidence that the system meets its requirements, and any literature using these terms needs to be read with this qualification in mind. To determine trustworthiness, we focus on methodologies and metrics that allow us to measure the degree of confidence that we can place in the entity under consideration. A different term captures this notion.

Definition 18-2. Security assurance, or simply assurance, is confidence that an entity meets its security requirements, based on specific evidence provided by the application of assurance techniques.

"What Goes Wrong"

According to MGraw and Viega [McGraw 03]:[13]

Developers like to know what's going on in their programs, especially when a program does something unforeseen, like cause an error. For this reason, some coders tend to put diagnostic information about errors directly in error messages displayed to the user. Attackers can and will use this information to exploit your code. In fact, clever attackers combine trusted input risks with blabbermouth error-reporting to exploit software. Put diagnostic information that could be used in an exploit somewhere safe (like an error log file), instead.

References

[Bishop 03]	Bishop, Matt. *Computer Security: Art and Science*. Boston, MA: Addison-Wesley, 2003.
[Howard 02]	Howard, Michael & LeBlanc, David. *Writing Secure Code, 2nd ed*. Redmond, WA: Microsoft Press, 2002.
[McGraw 03]	McGraw, Gary & Viega, John. "Keep It Simple." *Software Development.* CMP Media LLC, May, 2003.
[NIST 01]	NIST. *Engineering Principles for Information Technology Security*. Special Publication 800-27. US Department of Commerce, National Institute of Standards and Technology, 2001.
[Viega 02]	Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.

13. All rights reserved. It is reprinted with permission from CMP Media LLC.

Cigital, Inc. Copyright

Copyright © Cigital, Inc. 2005-2007. Cigital retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

For information regarding external or commercial use of copyrighted materials owned by Cigital, including information about “Fair Use,” contact Cigital at copyright@cigital.com[1].

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com